Diskrete Fouriertransformation

- Motivation: Gleichung der schwingenden Saite
- Crashkurs “Komplexe Zahlen”
- Anwendung: Mustererkennung, Signalverarbeitung (skizziert)
- Anwendung: Multiplikation von Polynomen
- Schnelle Implementierung mit *divide-and-conquer* (FFT)

Diskrete Fouriertransformation

Eine Saite der Länge 1 sei an ihren Enden fest eingespannt. Ihre Auslenkung an der Stelle x (wobei $0 \leq x \leq 1$) zum Zeitpunkt t (wobei $t \geq 0$) bezeichnen wir mit $u(x, t)$.
Unter vereinfachenden Annahmen gilt:

$$c^2 \frac{\partial^2}{\partial x^2} u(x,t) = \frac{\partial^2}{\partial t^2} u(x,t)$$

- c ist eine Konstante, in die Spannung und Masse der Saite eingehen.
- Die linke Seite entspricht der Krümmung der Saite an der Stelle x zum Zeitpunkt t; daraus resultiert eine Rückstellkraft.
- Die rechte Seite entspricht der Beschleunigung, also nichts anderes als Newtons Gesetz "Kraft ist Masse mal Beschleunigung".

Theorie der partiellen DGL $\Rightarrow$: ist $u(x,0)$ vorgegeben (Anfangsauslenkung), gilt $\frac{\partial u}{\partial t}(x,0) = 0$ (Saite wird nicht angeschoben) und gilt $u(0,t) = u(1,t) = 0$ (Saite ist an den Enden eingespannt), so ist $u(x,t)$ für alle $0 \leq x \leq 1$ und $t \geq 0$ eindeutig festgelegt (entsprechend der Anschauung!)
Aber wie rechnet man $u(x,t)$ aus $v(x) := u(x,0)$ aus??
Dieses Problem löste FOURIER im 19.Jh.

Fouriers Lösung

Wenn $v(x) = u(x, 0) = \sin(\pi k x)$, dann ist
$u(x, t) = \sin(\pi k x) \cos(\pi c k t)$.
Beweis durch Nachrechnen:

$$c^2 \frac{\partial^2}{\partial x^2} u(x, t) = -c^2 \pi^2 k^2 u(x, t) = \frac{\partial^2}{\partial t^2} u(x, t)$$

Falls $v(x) = \sum_{k=1}^{\infty} \alpha_k \sin(\pi k x)$ dann
$u(x, t) = \sum_{k=1}^{\infty} c_k \sin(\pi k x) \cos(\pi c k t)$.
Fouriers geniale Feststellung: Jedes stetige v ist von dieser Form:

$$\alpha_k = \frac{2}{\pi k} \int_0^{\pi} v(x) \sin(\pi k x) \mathrm{d}x$$

Die Folge α_k ist die *Fouriertransformation* von v (genauer: Folge der Fourierkoeffizienten).

Diskrete Fouriertransformation

Gegeben sei eine Folge $a_0, a_1, \ldots, a_{n-1}$ reeller Zahlen. Z.B. Stützstellen einer Funktion.
Die *diskrete Fouriertransformation* (DFT) dieser Folge ist die Folge $y_0, y_1, \ldots, y_{n-1}$ *komplexer Zahlen* definiert durch

$$y_k = \sum_{j=0}^{n-1} e^{\frac{2\pi i}{n} jk} a_j$$

Zusammenhang mit Motivation: $e^{i\phi} = \cos(\phi) + i \sin(\phi)$ (EULERsche Formel). Die α_k können aus Imaginärteil der DFT näherungsweise bestimmt werden.
Wir interessieren uns für die DFT als eigenständige Operation aus anderen (aber verwandten) Gründen.

Komplexe Zahlen

Menge der komplexen Zahlen $\mathbb{C}$ entsteht aus $\mathbb{R}$ durch formale Hinzunahme von $i = \sqrt{-1}$, also $i^2 = -1$.

Eine komplexe Zahl ist ein Ausdruck der Form $a + ib$ wobei $a, b \in \mathbb{R}$.

Addition: $(a_1 + ib_1) + (a_2 + ib_2) = (a_1 + a_2) + i(b_1 + b_2)$.

Multiplikation: $(a_1 + ib_1)(a_2 + ib_2) =$
$a_1a_2 + i(a_1b_2 + a_2b_1) + i^2b_1b_2 = (a_1a_2 - b_1b_2) + i(a_1b_2 + a_2b_1)$.

Division: $\frac{a_1+ib_1}{a_2+ib_2} = \frac{(a_1+ib_1)(a_2-ib_2)}{(a_2+ib_2)(a_2-ib_2)} = \frac{a_1a_2+b_1b_2+i(a_2b_1-a_1b_2)}{a_2^2+b_2^2}$

Bemerkung: a ist der Realteil der komplexen Zahl $a + ib$, die Zahl b ist ihr Imaginärteil. Man schreibt $a = \mathrm{Re}(a + ib)$ und $b = \mathrm{Im}(a + ib)$.

Zahlenebene

Man visualisiert komplexe Zahlen als Punkte oder Ortsvektoren.
$a + ib$ entspricht dem Punkt (a, b).
Addition entspricht dann der Vektoraddition.
Betrag: $|a + ib| = \sqrt{a^2 + b^2}$. Es gilt $|wz| = |w||z|$.
Man kann einer komplexen Zahl $z = a + ib$ ihren Winkel im Gegenuhrzeigersinn von der reellen Achse aus gemessen, zuordnen.
Dieser Winkel ϕ heißt Argument von z, i.Z. $\arg(z)$. Es gilt $\phi = \texttt{atan2}(a, b)$. D.h. $\tan(\phi) = b/a$ und $b \leq 0 \Leftrightarrow |\phi| \geq \pi/2$.
Aus Betrag $r = |z|$ und Argument $\phi = \arg(z)$ kann man z rekonstruieren:

$$z = r\cos(\phi) + ir\sin(\phi)$$

Mit $e^{i\phi} = \cos(\phi) + i\sin(\phi)$ erhält man

$$z = re^{i\phi}$$

Dies ist die Polarform der komplexen Zahl z.
Es gilt: $(r_1 e^{i\phi_1})(r_2 e^{i\phi_2}) = r_1 r_2 e^{i(\phi_1+\phi_2)}$.
Beim Multiplizieren multiplizieren sich die Beträge und addieren sich die Argumente.

Einheitswurzeln

Sei $n \in \mathbb{N}$. Wir schreiben $\omega_n := e^{2\pi i/n}$.
Es gilt $\omega_n^n = 1$ und allgemeiner $(\omega_n^k)^n = 1$.
Die n (verschiedenen) Zahlen $1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}$ heißen daher n-te Einheitswurzeln.
Sie liegen auf den Ecken eines regelmäßigen n-Ecks mit Mittelpunkt Null und einer Ecke bei der Eins.
Satz: Es gilt für $k = 0, \ldots, n-1$:

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = \begin{cases} n, \text{ falls } k = 0 \\ 0, \text{sonst} \end{cases}$$

Beweis mit geometrischer Reihe oder durch Veranschaulichung in der Zahlenebene.
Satz: Ist n gerade, so sind die $n/2$-ten Einheitswurzeln gerade die Quadrate der n-ten Einheitswurzeln.
Beweis: Eine Richtung: $((\omega_n^k)^2)^{n/2} = 1$. Andere Richtung: $\omega_{n/2}^k = (\omega_n^2)^k$.

Einheitswurzeln und DFT

Definition der DFT: $y_k = \sum_{j=0}^{n-1} e^{\frac{2\pi i}{n} jk} a_j$.

Es ist $y_k = \sum_{j=0}^{n-1} \omega_n^{jk} a_j$.

Die DFT ergibt sich also als die Folge
$A(1), A(\omega_n), A(\omega_n^2), \ldots, A(\omega_n^{n-1})$ wobei $A(x) = \sum_{j=0}^{n-1} a_j x^j$.

Translationsinvarianz

Satz: Sei $(a_j)_{j=0}^{n-1}$ eine Folge reeller Zahlen und $b_j = a_{(j+d) \bmod n}$. Sei $(y_k)_k$ die DFT von $(a_j)_j$ und $(z_k)_k$ die DFT von $(b_j)_j$. Es gilt $|y_k| = |z_k|$. D.h. der Betrag der DFT, das sog. Intensitätsspektrum, ist translationsinvariant.

Beweis: $z_k = \sum_j \omega_n^{jk} a_{j+d \bmod n} = \sum_{j'} \omega_n^{(j'-d)k} a_{j'k} = \omega_n^{-dk} y_k$. Und $|\omega_n^x| = 1$.

Anwendung: Um festzustellen, ob ein bestimmtes Muster irgendwo in der Folge a vorhanden ist, vergleiche man die DFT von a mit der DFT des Musters. Insbesondere interessant bei 2D-DFT. Z.B. Schrifterkennung.

Inverse DFT

Gegeben sei eine Folge $y_0, \ldots, y_{n-1}$ komplexer Zahlen.
Die inverse DFT ist die Folge $a_0, \ldots, a_{n-1}$ definiert durch

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{-jk} y_k$$

Satz: Die inverse DFT ist tatsächlich die zur DFT inverse Operation.
Beweis: Wir verwenden das Kroneckersymbol
$\delta_{jl} = \texttt{if } j = l \texttt{ then 1 else 0}$.
Es genügt, die Behauptung für $a_j = \delta_{jl}$ für alle $l = 0, \ldots, n-1$ zu zeigen, da DFT, und inverse DFT linear sind und diese a's eine Basis bilden.
Sei also $a_j = \delta_{jl}$. Anwendung der DFT liefert $y_k = \omega_n^{kl}$. Anwendung der inversen DFT liefert dann

$$\frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{-jk} \omega_n^{kl} = \frac{1}{n} \sum_{k=0}^{n-1} (\omega_n^k)^{l-j} = \delta_{lj}$$

DFT und Polynome

Ein Polynom mit Gradschranke n ist ein formaler Ausdruck der Form

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

Die a_j heißen Koeffizienten des Polynoms A.
Man kann ein Polynom als Funktion auffassen (Polynomfunktion).
Z.B.: $A(x) = 1 + 2x + 3x^2$ und $A(2) = 17$.

Hornerschema

Sei $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$ ein Polynom mit Gradschranke n und x_0 eine Zahl.
Bestimmung von $A(x_0)$ nach HORNER mit $\Theta(n)$ Rechenoperationen:

$$A(x_0) = a_0 + x_0(a_1 + x_1(a_2 + \cdots + x_0(a_{n-2} + x_0 a_{n-1}) \ldots)$$

Interpolation

Ein Polynom ist durch seine Polynomfunktion eindeutig bestimmt, d.h., man kann die Koeffizienten aus der Polynomfunktion ausrechnen.
Genauer: Sei $A(x_k) = y_k$ für n verschiedene Stützstellen $x_0, \ldots, x_{n-1}$.
Dann gilt nach LAGRANGE:

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

Ausmultiplizieren liefert das gewünschte Polynom.
Bemerkung: Dies gilt nur bei den reellen Zahlen, oder genauer bei Körpern der Charakteristik 0. Im zweielementigen Körper induzieren x^2 und x dieselben Polynomfunktionen.

Addition und Multiplikation von Polynomen

Polynome kann man addieren und multiplizieren:
Die Summe zweier Polynome $A(x) = \sum_{j=0}^{n-1} a_j x^j$ und
$B(x) = \sum_{j=0}^{n-1} b_j x^j$ mit Gradschranke n ist das Polynom
$C(x) = \sum_{j=0}^{n-1} c_j x^j$ wobei $c_j = a_j + b_j$.
Das Produkt zweier Polynome $A(x) = \sum_{j=0}^{n-1} a_j x^j$ und
$B(x) = \sum_{j=0}^{n-1} b_j x^j$ mit Gradschranke n ist das Polynom
$C(x) = \sum_{j=0}^{2n-1} c_j x^j$ mit Gradschranke $2n$ wobei $c_j = \sum_{l=0}^{j} a_l b_{j-l}$.
Z.B.:
$(1 + 2x + 3x^2)(4 + 5x + 6x^2) = 1 + 13x + 25x^2 + 27x^3 + 18x^4$.
Sind Polynome als Koeffizientenliste repräentiert, so erfordert das Ausrechnen der Summe $O(n)$ Operationen; das Ausrechnen des Produkts erfordert $O(n^2)$ Operationen.

Punkt-Wert Repräsentation

Man kann Polynome auch als Punkt-Wert Liste repräsentieren, also als Liste von Paaren

$$((x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}))$$

wobei die x_k paarweise verschieden sind.
Z.B. $A(x) = 1 + 2x + 3x^2$ repräsentiert durch
$((0, 1), (1, 6), (-1, 2))$.

Punkt-Wert Repräsentation

Sind $((x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}))$
und $((x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1}))$
zwei solche Repräsentationen zu denselben Stützstellen, so erhält man Repräsentationen für Summe und Produkt als

$$((x_0, y_0 + y'_0), (x_1, y_1 + y'_1), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1}))$$

und

$$((x_0, y_0 y'_0), (x_1, y_1 y'_1), \ldots, (x_{n-1}, y_{n-1} y'_{n-1}))$$

D.h. Addition und Multiplikation von Polynomen in Punkt-Wert Repräsentation erfordert $O(n)$ Operationen.
Aber die Auswertung von Polynomen in Punkt-Wert Repräsentation erfordert $O(n^2)$ Operationen.

Multiplikation mit DFT

Erinnerung: Die DFT einer Liste $(a_0, \ldots, a_{n-1})$ ist die Liste $(y_0, \ldots, y_{n-1})$ wobei

$$y_k = A(\omega_n^k)$$

und

$$A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

Die DFT von $(a_0, \ldots, a_{n-1})$ ist die Punkt-Wert Repräsentation des Polynoms $\sum_{j=0}^{n-1} a_j x^j$ zu den Stützstellen $\omega_n^0, \ldots, \omega_n^{n-1}$, also den n-ten Einheitswurzeln.

Um zwei Polynome in Koeffizientenform zu multiplizieren kann man ihre DFTs punktweise multiplizieren und dann die inverse DFT anwenden.

Multiplikation mit DFT

- Gegeben zwei Polynome A, B mit Gradschranke n als Koeffizientenlisten
- Fülle mit Nullen auf um formell Polynome mit Gradschranke $2n$ zu erhalten.

$$a = (a_0, \ldots, a_{2n-1}), \quad b = (b_0, \ldots, b_{2n-1})$$

- Bilde jeweils DFT der beiden expandierten Koeffizientenlisten: $(y_0, \ldots, y_{2n-1}) = \mathrm{DFT}_{2n}(a)$, $(z_0, \ldots, z_{2n-1}) = \mathrm{DFT}_{2n}(b)$,
- Multipliziere die DFTs punktweise: $w_k = y_k z_k$ für $k = 0, \ldots, 2n-1$,
- Transformiere zurück um Koeffizientenliste von $C = AB$ zu erhalten: $(c_0, \ldots, c_{2n-1}) = \mathrm{DFT}^{-1}(w_0, \ldots, w_{2n-1})$.

Leider braucht das Ausrechnen der DFT $\Theta(n^2)$ Rechenoperationen, sodass keine unmittelbare Verbesserung eintritt.
Mit *divide and conquer* kann man aber die DFT in $\Theta(n \log n)$ ausrechnen!

Schnelle Fouriertransformation (FFT)

Sei n bis auf weiteres eine Zweierpotenz.
$A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$.
Es ist $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$ wobei

$$A^{[0]}(z) = a_0 + a_2 z + a_4 z^2 + \cdots + a_{n-2} z^{n/2-1}$$

$$A^{[1]}(z) = a_1 + a_3 z + a_5 z^2 + \cdots + a_{n-1} z^{n/2-1}$$

Beispiel:

$$A(x) = 1 + 2x + 3x^2 + 4x^3$$
$$A^{[0]}(z) = 1 + 3z$$
$$A^{[1]}(z) = 2 + 4z$$

Schnelle Fouriertransformation (FFT)

Also gilt unter Beachtung von $\omega_n^{2k} = \omega_{n/2}^k$:

$$A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + \omega_n^k A^{[1]}(\omega_{n/2}^k)$$

Für $k < n/2$ sind die beiden rechtsstehenden Polynomauswertungen selbst DFTs, aber zur halben Größe.
Für $k \geq n/2$ passiert nichts neues, da $\omega_{n/2}^{k+n/2} = \omega_{n/2}^k$.
Nur der Vorfaktor ω_n^k ändert sich: $\omega_n^{k+n/2} = -\omega_n^k$.

Schnelle Fouriertransformation (FFT)

```
RECURSIVE-FFT(a)
1   n ← length[a]
2   if n = 1
3         then return a
4   ω_n ← e^{2πi/n}
5   ω ← 1
6   a^[0] ← (a_0, a_2, ..., a_{n/2-2})
7   a^[1] ← (a_1, a_3, ..., a_{n/2-1})
8   y^[0] ← RECURSIVE-FFT(a_0, a_2, ..., a_{n/2-2})
9   y^[1] ← RECURSIVE-FFT(a_1, a_3, ..., a_{n/2-1})
10  for k ← n/2 - 1 do
11        y_k ← y_k^[0] + ω y_k^[1]
12        y_{k+n/2} ← y_k^[0] - ω y_k^[1]
13        ω ← ω ω_n
14  return y
```

Schnelle Fouriertransformation (FFT)

- Nach dem Vorhergesagten ist klar, dass y die DFT von a ist.
- Die Laufzeit $T(n)$ von RECURSIVE-FFT erfüllt $T(n) = 2T(n) + \Theta(n)$, also $T(n) = \Theta(n \log n)$.
- In der Praxis rechnet man die DFT iterativ von unten her aus (vgl. dynamische Programmierung), indem man sich überlegt für welche Arrays rekursive Aufrufe stattfinden.
- Man kann die FFT auch gut parallelisieren.
- Zur Multiplikation von Polynomen mit ganzen Koeffizienten empfiehlt sich die modulare DFT.

DFT in 2D

Sei $a_{jk}, j = 0, \ldots, n-1, k = 0, \ldots, n-1$ eine Matrix von Zahlen. Die DFT von a ist die Matrix
$y_{uv}, u = 0, \ldots, n-1, v = 0, \ldots, n-1$ definiert durch

$$y_{uv} = \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \omega_{2n}^{ju+kv} a_{jk}$$

Translationsinvarianz des Intensitätsspektrums gilt jetzt bezüglich Verschiebung in j und k Richtung.

Schnelle inverse DFT

Erinnerung: Die inverse DFT einer Liste $(y_0, \ldots, y_{n-1})$ ist die Liste $(a_0, \ldots, a_{n-1})$ wobei

$$a_j = \frac{1}{n} Y(\omega_n^{-j})$$

und

$$Y(x) = y_0 + y_1 x + \cdots + y_{n-1} x^{n-1}$$

Aber $\omega_n^{-j} = \omega_n^{n-j}$. Man erhält die inverse DFT von y aus der DFT von y durch Umgruppieren und Dividieren durch n:

Schnelle inverse DFT

INVERSE-FFT(y)

1 $n \leftarrow \mathit{length}[y]$
2 $a \leftarrow$ RECURSIVE-FFT(y)
3 **for** $j \leftarrow 1$ **to** $n/2 - 1$ **do**
4 $\quad$ *Swap* $a_j \leftrightarrow a_{n-j}$
5 **for** $j \leftarrow 0$ **to** $n - 1$ **do**
6 $\quad$ $a_j \leftarrow a_j / n$
7 **return** a

Faltung

Seien $a = (a_0, \ldots, a_{n-1})$ und $b = (b_0, \ldots, b_{n-1})$ zwei Folgen von je $2n$ Zahlen wobei $a_k = b_k = 0$ für $k \geq n$.
Die Faltung (engl.: *convolution*) $a \star b$ ist die Folge von $2n$ Zahlen definiert durch

$$(a \star b)_k = \sum_{j=0}^{k} a_j b_{k-j}$$

Beachte: $a \star b$ ist die Koeffizientenfolge des Produkts der durch a und b repräsentierten Polynome.
Satz: $a \star b = \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(a)\mathrm{DFT}_{2n}(b))$ wobei das Produkt komponentenweise zu verstehen ist.

Anwendung der Faltung

Faltung mit $b = (-1, 2, -1, 0, 0, 0, 0, \ldots, 0)$ hebt Sprünge hervor.
In 2D: *edge detection*.
Faltung mit $b = (1, 2, 1, 0, 0, 0, 0, 0, \ldots, 0)$ verwischt.
In 2D: *blurring*.
Die Operation $c \mapsto \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(c)/\mathrm{DFT}_{2n}(b))$ macht die Faltung mit b rückgängig.
In 2D: *sharpening*
Die Faltung hat auch zahlreiche Anwendungen bei der Audiosignalverarbeitung.